The ASSIST Programming Environment

Massimo Coppola

14/05/2010 – Computer Science & Networking, SPD Course
With contributions from the whole Research Group on
Parallel Architectures at Dip. di Informatica, Univ. di Pisa

Summary

- The ASSIST skeleton system
- Basic Concepts
  - Code encapsulation
  - Execution
- Syntax and Semantics
  - The ASSIST Constructs
  - Parallelism expression
- Advanced features
  - Run-time, Dynamic Adaptivity
  - Heterogeneous Platforms
  - Component orientation
- Future Extensions

The ASSIST skeleton system

- Skeleton-based parallel programming environment
  - Compilation, deployment, execution
  - Targets scientific and industrial needs
    - High performance
    - Programmability, portability, interoperability, time-to-market,
- Developed with ideas from other prototypes
  - P3L (classical skeletons, C -based implementation)
  - SkIE (classical skeletons, multi language)
  - Muskel (data-flow Java based)
- Paradigm shift: “modable” skeletons, “escapes”

The ASSIST skeleton system

- Separation of concerns
  - Application programmer vs. system programmer
- Expressive power
  - Most current patterns available through parmod
- Code reuse
  - C, C++, F77 (Java)
- External objects/library access support
  - Provide escapes to unstructured / external resources
- Layered implementation
  - Compiler, deploy tools, multitarget run time
- Multiple target architectures
  - Different CPUs/Memory & different Operating Systems

Basic Concepts

- Data flow-interaction
  - Typed streams
- Code encapsulation
  - Well defined interfaces around code modules
- Which parallel skeletons?
  - Flexible, extendable approach
- Execution
- Deployment tools
- Adaptivity

The ASSIST Constructs

- Streams
  - main interaction mechanism
  - complemented by shared memory data structures
- Seq
  - the simplest case of a code module
  - multi language code encapsulation
- Generic
  - pipeline, DAG, generic graph task parallelism
- Parmod
  - Multiple-pattern parallel skeleton

Streams and data types

- Interaction mechanism among modules
- Typed (stream packets are ASSIST types)
- ASSIST types = CORBA types
  - C-like syntax
  - serializable data structures
  - predefined inter-language equivalence
  - technology ages fast…
- Stream management within the run-time
  - Implementation details are hidden
  - exploit binary or XDR formats

Sequential code modules

- Seq defines a simple code module
  - To run sequentially on any single computing resource
  - With specified interface: type of input and output parameters
  - Code defined by a **proc** section
- Proc specifies sequential code behavior
  - General use in ASSIST (also within parallel modules)
  - **Which** language
  - What is the actual code, and what are its interfaces

The *seq* and *proc* Constructs

```
My_seq_module(    input_stream long x
                  output_stream long y)
  { f(in x out y); }

proc f(in long a out long b)
  inc<“myHeader.cpp”,
      “mySource.cpp”>
  path<“/home/marcod/myIncludes”>
  obj<“myObjectCode.o”>
  src<“mySource.c”>
$c++{  /* here goes your code…*/ }c++$
```

can include/link:
source, headers, externally generated object-code files and sequential libraries
also different source languages

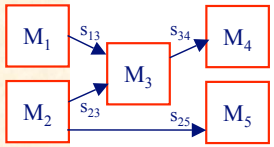
The *generic* Construct

- Short for generic graph
  - can have loops
  - pipeline and direct acyclic graphs (DAG) as special cases
- Allows to define unconstrained data-flow graphs of modules
  - Sequential, parallel modules and nested generic
- Each module represented by its functional interfaces
  - input and output streams
  - Multiple inputs and outputs: supports nondeterministic behavior
- Data-flow + shared status
  - Unsynchronized shared var.s (rely on program structure!)

Generic example

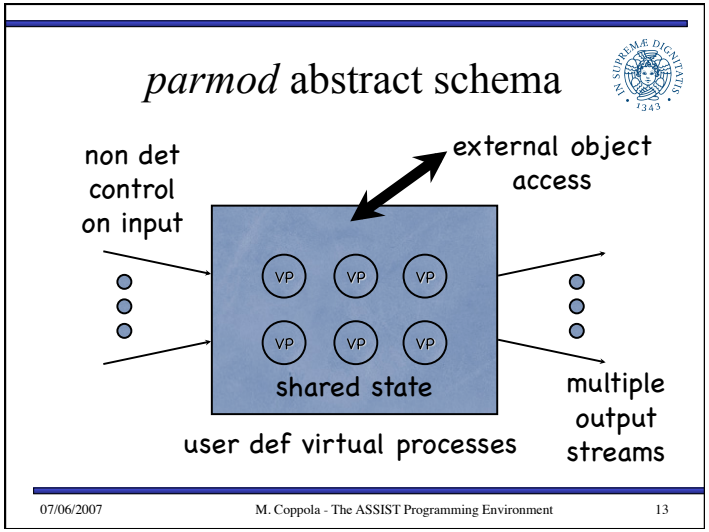
```
generic main (. . .)
{
    stream   int s13;
    stream   int [N][N] s23;
    stream   int [N][N] s34;
    stream   int s25;

   M1 (ouput_stream s13);
   M2 (output_stream s23, s25);
   M3    (input_stream    s13,    s23;
   output_stream s34);
   M4 (input_stream s34);
   M5 (input_stream s25);
}
```



parmod = generic PARallel MODule

- structured way of defining parallel computations
- abstracting away from actual mechanisms
  - logical parallel activities
  - logical data sharing
  - specification of cooperation with the “outside”
- syntax special cases : farm, map, ….
- + expressiveness = deal with special cases
  - classical skeletons are enough, usually
  - mix skeleton behaviors / switch among them


parmod abstract schema
non det
control
on input
external object
access
VP
VP
VP
VP
VP
VP
shared state
user def virtual processes
multiple
output
streams
07/06/2007
M. Coppola - The ASSIST Programming Environment
13